Chapter

9

Macros: An Overview

A *macro* is a list of actions you want ReportSmith to carry out for you. Macros let you enhance your productivity and add complex features to your reports. Use one of the macros ReportSmith provides for you or create a custom-tailored macro to do any of the following tasks (and many more):

- Automatically load and print reports.
- Customize ReportSmith by creating a custom dialog box, hiding or disabling menu items, and creating new menu items.
- Create derived fields.
- Perform complex calculations.

This chapter defines macros, shows you how to create and use them to simplify many ReportSmith tasks, and describes some advanced macro concepts.

Introduction to macros

In this introductory section, you'll see what macros are and how to use them. You'll also see the difference between global and local macros and how they are used. This section also provides basic information about using macros with other Windows applications and about the ReportBasic programming language.

What are macros?

Macros are small programs that you build using the ReportBasic programming language. You do this by creating a macro (writing ReportBasic commands), and then linking it to:

- a ReportSmith event such as opening a report.
- a computer-system event such as opening ReportSmith.
- a user-generated event such as pressing a particular key.

What is ReportBasic?

ReportBasic is a specially customized version of the BASIC programming language, designed to provide ReportSmith users with a complete, high-level programming utility. Because it is a complete programming language, ReportBasic can be used for a variety of tasks, including:

- **Calculating derived fields**

  With ReportBasic, you can create derived fields that are not stored in the database but are a result of a calculation using the values stored in your report fields.

- **Conditional formatting and conditional text values**

  You can use ReportBasic to highlight a value if it meets certain criteria in a report. For example, you might want to display all sales transactions in a report that are greater than a stated quota in a bolder, larger font than those that fall below the quota. ReportBasic can also be used to warn a report reader that invalid data exists in a database.

- **Customized user interface**

  You can use ReportBasic to create custom user interfaces such as dialog boxes that contain user-entry fields, command buttons, and check boxes. ReportBasic can also be used to add, remove, or disable ReportSmith menus, menu items, and toolbar icons. A report designer may want to do this to prevent a user from modifying a report or to add specific menu items that run specific macros.

Macro scope

You can create two kinds of macros: a global macro or a report macro. A macro linked to ReportSmith itself, that runs each time you run ReportSmith, is called a *global macro*, or is said to have *global scope*. A *report macro*, or one that has only *report-specific scope*, is linked to a particular report and becomes a part of that report; it does not apply to and is unavailable from any other report. When you save the report, ReportSmith saves any associated report macros in the *.RPT* file.

Either of these macro types can also be an *inter-application* macro, designed to communicate in some way between ReportSmith and another application.

Global macros

You can create global macros to customize ReportSmith or to automate global tasks. For example, suppose you want to combine loading, printing, and closing a report into one step. This same macro can also display a dialog box that prompts the user for the report names. When you run the macro, it opens, prints, and closes each report the user specifies in the dialog box.

You can create a global macro that prompts for a password, letting only authorized people access a given report, or one that hides the toolbar or executes a specific menu command, such as File|Open.

Report macros

A r*eport macro* automates tasks that are *specific to a particular report*. For example, you can create a report macro that only lets a certain user access a confidential report by prompting for a password or by verifying the value of a DOS variable. For example, you might have this line in your AUTOEXEC.BAT:

```
SET USER=John Doe
```

In this case, the macro opens the report only if the environment variable is set to John Doe.

```
  ' This code will abort the open event if the user is not John Doe
  ' Get the user name
USER$=ENVIRON$("USER")
IF USER$ <> "John Doe" THEN
  ' Abort the report's loading
RESUMEEVENT 0
END IF
```

You can create a report macro that behaves dynamically according to changes in the information it receives. For example, suppose you have a report that is unusually large, and you don’t want users opening it when network traffic is high (usually between 4:00 and 5:00 every weekday). You can prevent a user from opening the report during peak time and display a message explaining why. In this case, the macro behaves differently according to the time of day.

You also can create a report macro that steps through the fields in a report column and produces a summary field. For example, suppose your Sales Department has a commission structure with different multipliers for different quotas. You could write a macro that calculates a sales person's commission based on the commission structure and then place that value into your report.

Inter-application macros

Report macros let you extend to other Windows applications. For example, if you have a report with a column listing part numbers and an Excel spreadsheet with a table listing these part numbers and their corresponding part names, you can write a macro that changes the part number to the part name in your report.

You can also add C-language (or dialects such as C++) functions from a Windows Dynamic Link Library (DLL) into a report macro. If you have a Windows DLL containing functions that perform complicated financial computations, you can create a report macro that uses those functions to calculate the data in your report.

Creating and working with macros

This section shows you how to create a macro. It takes you step-by-step through the creation of a simple macro and shows you how to link a macro to objects and events.

First, you create a macro that displays a message box. Then you link the macro to an object and an event.

Creating a simple macro

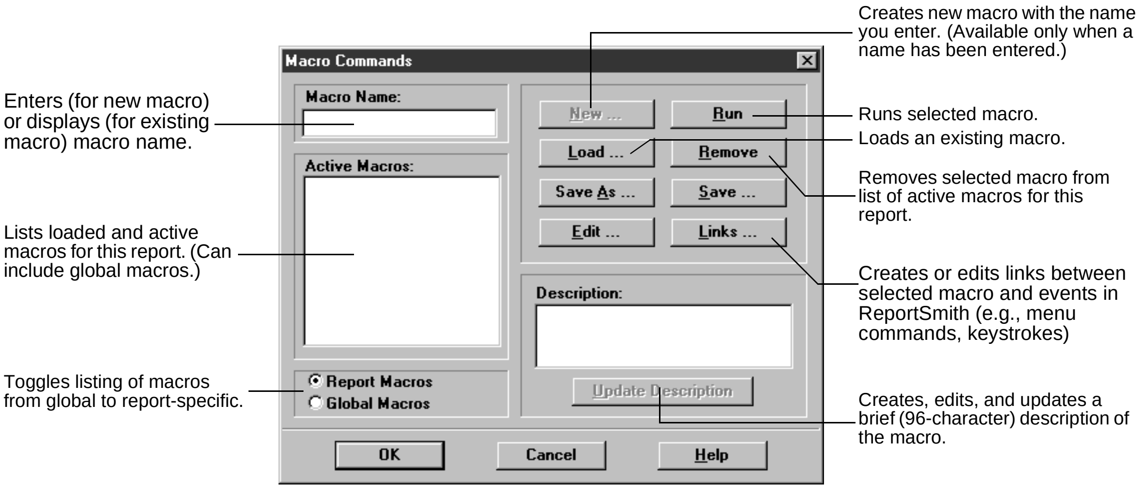
You create a ReportBasic macro in much the same way you would create an SQL query to build a report. First, access a list of available macros to determine whether you can modify an existing macro to suit your needs, and then edit or create the macro commands you need.

■ To create a macro,

1 Select Tools|Macro.

The Macro Commands dialog box appears, as in Figure 9.1. Note that, by default, ReportSmith assumes you are creating a global macro and activates the Global Macros radio button at the bottom left-hand corner. Also note that when you select Tools|Macro with a report open, ReportSmith enables the Report Macros radio button for you to select if you are creating a report macro.

Figure 9.1: The Macro Commands dialog box)


Macro Commands
Macro Name:
Active Macros:
Report Macros
Global Macros
New ...
Run
Load ...
Remove
Save As ...
Save ...
Edit ...
Links ...
Description:
Update Description
OK
Cancel
Help
Creates new macro with the name you enter. (Available only when a name has been entered.)
Enters (for new macro) or displays (for existing macro) macro name.
Runs selected macro.
Loads an existing macro.
Removes selected macro from list of active macros for this report.
Lists loaded and active macros for this report. (Can include global macros.)
Creates or edits links between selected macro and events in ReportSmith (e.g., menu commands, keystrokes)
Toggles listing of macros from global to report-specific.
Creates, edits, and updates a brief (96-character) description of the macro.

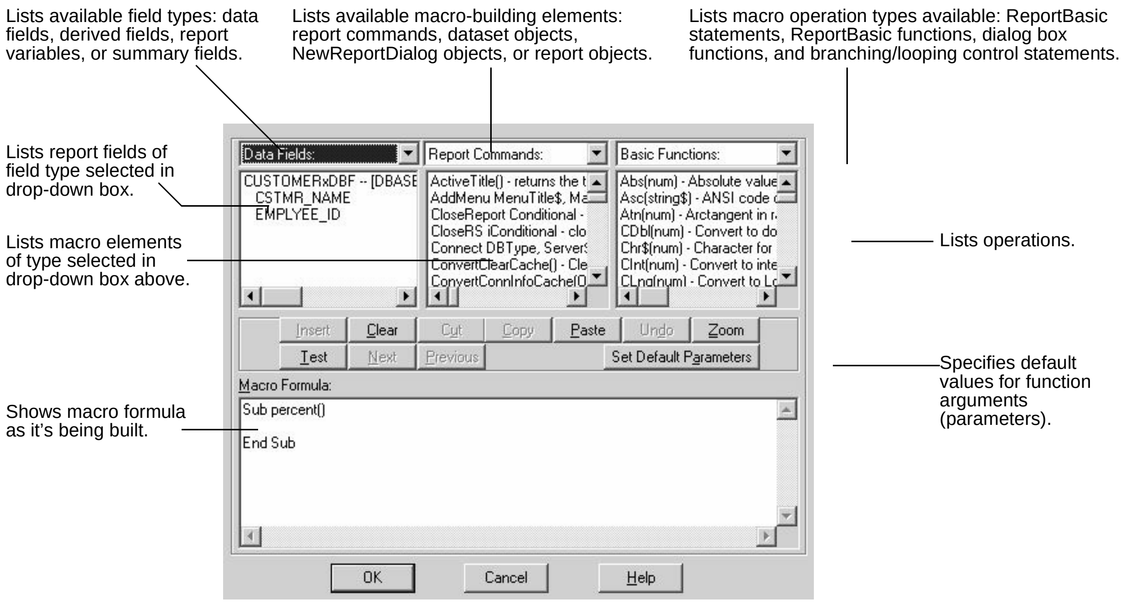
2 In the Macro Name text box, enter the name of your new macro. You cannot choose a macro name that duplicates a BASIC-language reserved word. This results in errors, and your macro will not function properly.

Note that ReportSmith enables the New button when you enter a macro name in the Macro Name text box; it enables the other macro command buttons after you have created the macro.

3 Choose New.

The Edit Macro dialog box appears, as in Figure 9.2Figure 9.2. (ReportSmith commands and the Basic commands are listed and described in “The Macro Command Reference” in online Help.)

Figure 9.2: The Edit Macro dialog box


Lists available field types: data fields, derived fields, report variables, or summary fields.
Lists available macro-building elements: report commands, dataset objects, NewReportDialog objects, or report objects.
Lists macro operation types available: ReportBasic statements, ReportBasic functions, dialog box functions, and branching/looping control statements.
Lists report fields of field type selected in drop-down box.
Lists macro elements of type selected in drop-down box above.
Lists operations.
Specifies default values for function arguments (parameters).
Shows macro formula as it's being built.
Data Fields:
Report Commands:
Basic Functions:
CUSTOMERxDBF -- [DBASE
CSTMR_NAME
EMPLYEE_ID
ActiveTitle() - returns the t
AddMenu MenuTitle$, Ma
CloseReport Conditional -
CloseRS iConditional - clo
Connect DBType, Server$
ConvertClearCache() - Cle
ConvertConnInfoCache(O
Abs(num) - Absolute value
Asc(string$) - ANSI code
Atn(num) - Arctangent in r
CDbl(num) - Convert to do
Chr$(num) - Character for
CInt(num) - Convert to inte
CLng(num) - Convert to Lo
Insert
Clear
Cut
Copy
Paste
Undo
Zoom
Test
Next
Previous
Set Default Parameters
Macro Formula:
OK
Cancel
Help

```
Sub percent()

End Sub
```

4 Drag the commands you want into the Macro Formula text box. The beginning and ending commands are automatically inserted for you by ReportSmith.

—or—

Type commands directly into the Macro Formula text box. The beginning and ending commands are automatically inserted for you by ReportSmith.

5 Choose Test.

If you entered the macro commands correctly, ReportSmith displays a success message. If your macro formula is incorrect, ReportSmith informs you of errors so you can fix the macro.

6 When you have compiled an error-free macro, choose OK to return to the Macro Commands dialog box and run the macro to test its effect.

7 In the Macro Commands dialog box, choose Run.

8 If your macro performs as intended, choose OK to close the Macro Commands dialog box; otherwise, choose Edit to return to the Edit Macro dialog box and modify your macro as appropriate.

Loading an existing macro

From time to time, you might find it easier and faster to modify the functionality of a pre-installed ReportSmith macro file or a macro file someone else has created, rather than create a new macro “from scratch.” You can do this by loading the macro file into the macro facility.

Note: You can load only those macros that have been saved to a .MAC file. See “Saving a macro to a .MAC file” on page 208 for more information on saving macros.

■ To load a macro file,

1 Select Tools|Macro to display the Macro Commands dialog box.

You cannot load a macro file that duplicates the name of a currently active macro (Check the “Active Macros” list.). If you attempt to do this, ReportSmith displays an error message prompting you to first remove the duplicate name from the list of active macros.

2 Press the Load button.

The Load Macro dialog box appears.

3 Double-click the macro file you want to load.

The macro you select becomes the active macro in the Macro Commands dialog box.

4 Choose Edit to edit the macro as necessary, and then choose OK to return to the Macro Commands dialog box.

5 Choose Save to update your macro file, and then choose OK to close the Macro Commands dialog box.

Linking a macro to an event

The next step in using a macro is to link it to a “trigger” event—a system event such as an elapsed time interval, a ReportSmith event such as opening a particular report, or a user event such as pressing a particular key. You select the event that triggers the macro by using the Macro Links dialog box. Refer to “Linking to events” on page 247 for detailed definitions of the types of events to which you can link a macro.

■ To link a macro to an event,

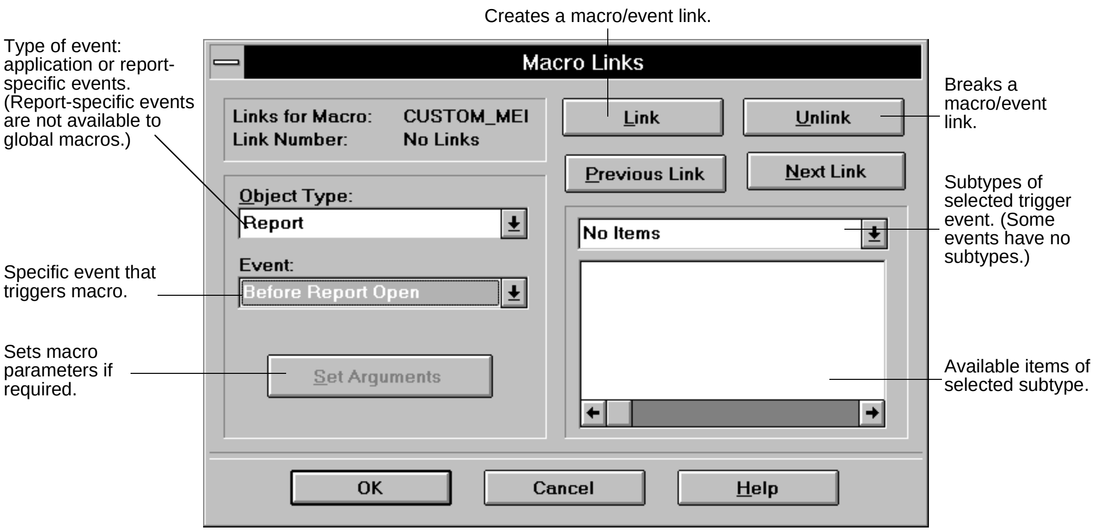
1 Choose Tools|Macro.

The Macro Commands dialog box appears, as in Figure 9.1 on page 205.

2 Select your macro and choose Links.

The Macro Links dialog box appears, as in Figure 9.3.

Figure 9.3: **The Macro Links dialog box**


Creates a macro/event link.
Type of event: application or report-specific events. (Report-specific events are not available to global macros.)
Breaks a macro/event link.
Subtypes of selected trigger event. (Some events have no subtypes.)
Specific event that triggers macro.
Sets macro parameters if required.
Available items of selected subtype.
Macro Links
Links for Macro: CUSTOM_MEI
Link Number: No Links
Link
Unlink
Previous Link
Next Link
Object Type:
Report
Event:
Before Report Open
No Items
Set Arguments
OK
Cancel
Help

3 Select an object type for the event. Global macros can be linked only to the Application object type while report-specific macros can be linked to a much wider variety of types.

4 Select an event. Depending on the event you select, subtypes and specific items will appear on the right side of the dialog box. Some events have no subtypes.

5 If appropriate, select a subtype and specific item to link to the macro. For example, the “Keystroke” event takes both the “Keys” subtype and a specific key selection.

6 Choose Link.

ReportSmith links the macro to the specified event. The Link Number display field (which originally displayed “No Links”) now displays “1 of 1”, indicating you have linked the macro to one event. If you link this macro to additional events, this display is updated to reflect both the number of total links and which link (within that total number) is currently open.

7 Choose OK to return to the Macro Commands dialog box.

Saving a macro to a .MAC file

ReportSmith lets you save a macro in a special-format (*.MAC) file. You can share the macro with other ReportSmith users by copying it to a disk or sending it over a network to another workstation. You may even want to copy it and use it in another report on your own system. You might want to save macros in this format for archiving.

Whether or not you save the macro to a .MAC file, ReportSmith saves it *within the macro facility*. You can access it by selecting Tools|Macro although you will not be able to distribute it to other users if it is not saved to a file. For report macros, ReportSmith saves the links with the report itself in the .RPT file. For global macros, ReportSmith saves the links with the ReportSmith application.

Note: When you save a macro in a .MAC file, you save *only the macro code itself*. Links to events and objects are not stored in the .MAC file since they might not be appropriate for other users.

- To save a macro to a .MAC file,

1 Choose Tools|Macro.

The Macro Commands dialog box appears, as in Figure 9.1 on page 205.

2 Select the macro you want to save.

3 Choose Save As.

4 Specify a name and location for the macro file, and then choose OK.

A macro file name need not be the same name as the macro itself. Characters, numbers, and underscores are allowed (no spaces), but the first character *must* be a number or letter.

5 Choose OK to close the Macro Commands dialog box.

Removing a macro

Occasionally, a macro might become obsolete or no longer needed. You can remove the macro from the list of active global or report-specific macros while still retaining it on your hard disk in case you need to reuse it later.

- To remove a macro from the list of active macros,

1 Choose Tools|Macro.

The Macro Commands dialog box appears.

2 In the Active Macros list box, select the macro you want to remove, and then choose Remove.

The macro no longer appears in the list of active macros.

3 Choose OK to close the Macro Commands dialog box.

Advanced macro concepts

This section discusses advanced concepts for ReportSmith’s macro facility, beginning with the different types of macro commands. Later in this section, you’ll see when and how to use command-line parameters and how to call other macros.

Macro command types

There are two types of macro commands: Functions and Statements.

- *Functions* return values.
- *Functions* must have their parameters surrounded by quotes.
- *Functions* use required parentheses around the list of parameters.
- *Statements* do not return a value.
- *Statements* have quotes around their parameters as well, but no parentheses.
- Some commands can be used as either functions or statements.

You use commands as either a function that requires a return value or a statement that does not require a return value. In combination, these command types gather and return values, and act upon them in a series of operations that, taken together, constitute your macro.

Commands as functions or statements

Although you can use many macro commands as either a function or a statement, some commands can be used *only* as a function, and others can be used *only* as a statement. A *function* must return a value, and its parameter must be within parentheses; a *statement* doesn’t use a return value and doesn’t use parentheses.

As an example of a command that can be used as either a statement or a function, consider the *MsgBox* command. Ordinarily, this command is used simply as a statement, and its parameters specify the message text and the icon (if any).

However, suppose that every time a user exits ReportSmith, you want a message to appear asking the user if he or she really wants to quit. Based on the user’s response to the message, the macro can either close ReportSmith or cancel the exit. In this case, you would use the *MsgBox* command as a function (rather than a statement) as in the following example:

Sub ConfirmExit()	This line names the macro “ConfirmExit”.
x = MsgBox ("Do you really want to Exit ReportSmith",4)	These parameters set the message text of the message box and specify that it should have a Yes button and a No button. Enclosing these parameters in parentheses makes this command a function, assigns the return value to “x”.
If x = 7 then	The “No” button returns a value of 7 when used in a function, so this statement means, “If the user chooses ‘No,’ then...”
ResumeEvent(0)	...do not resume the exit procedure, and...
End if	...quit checking the value of “x”.
End Sub	End of macro.

Using macro parameters

ReportSmith macros can accept *parameters*. A parameter lets you pass information from one macro to another and back again. By carefully using macro parameters, you can use one macro to call another.

■ To insert a parameter into a macro,

1 Select Tools|Macro.

The Macro Commands dialog box appears.

2 Choose Edit to edit the code for the macro you selected.

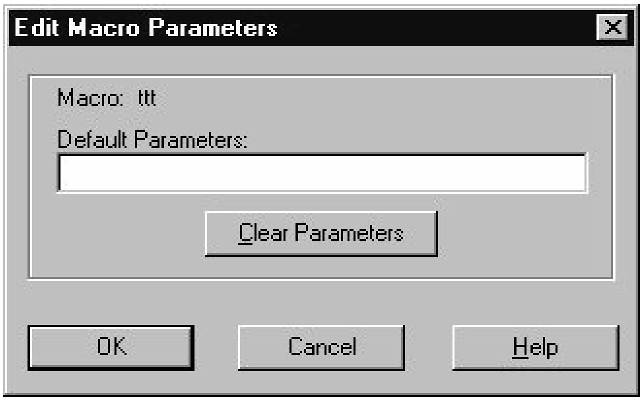
The Edit Macro dialog box appears.

3 Enter the macro parameters you want within the parentheses ( ) that follow the macro name. Separate the parameter variables with commas.

4 Choose Set Default Parameters.

The Edit Macro Parameters dialog box appears, as in Figure 9.4.

Figure 9.4: **Edit Macro Parameters dialog box**



5 In the Default Parameters text box, enter the default values (separated by commas) for the parameter variables in the same order as the variables appear in the macro code. Choose OK to return to the Edit Macro dialog box.

6 Choose OK to return to the Macro Commands dialog box, and then choose Run to run the macro now, or choose OK to return to ReportSmith.

Note: You can also use the RunMacro command to provide a string to specify the values for the parameter variables. If you specify a null string, ReportSmith uses the default parameters.

The following two macros show you how to pass parameters. In this case, you only have to place the parameter in the *called* macro. If you are going to pass it back again, you need to place it in the *calling* macro as well.

```
Sub calling()
' This variable allows you to pass along the double quotes and the value of the variable.
q$=chr$(34)
var1$=q$+ "It is Monday"+q$
RunMacro "called",var1$
End Sub

Sub called(var1$)
msgbox "Hey!"+var1$
End sub
```

Note: For this macro, make sure that you choose Set Default Parameters in the Edit Macro dialog box.

Parameters and events

When you link a macro to events, you can specify alternative values for the parameter variables in that macro for each individual event. If you choose not to supply alternative values, ReportSmith uses the default values you specified in the procedure above.

■ To supply values for macros (containing parameters) on a link-by-link basis,

1 Select your macro. In the Macro Links dialog box, choose Links.

2 Use the Previous Link and Next Link buttons to display the event for which you want to specify parameter variables.

3 Press the Set Parameters button to display the Edit Macro Parameters dialog box.

4 In the Default parameters text box, enter the values (separated by commas) for the parameter variables in the same order as the variables appear in the macro code. Press OK to return to the Macro Links dialog box. When you run the macro using the event you highlighted in step 4, ReportSmith will replace the default parameter values with the values you specified here.

5 To exit the macro facility, press OK in the Macro Links dialog box. Press OK in the Macro Commands dialog box.

Using the Creation event

The *creation* event applies to two objects: Group Header and Group Footer. When you link a macro to the Header/Footer creation event you must choose the grouping level of the header or footer to which you want to link the macro.

A macro linked to these events can call the *ResumeEvent* command with a parameter of 0 to suppress the creation of an individual group header or footer based on the data in the report.

For example, suppose you have a set of groups, and each has a number of records. If one of these groups has an insignificant number of items, the following macro would suppress the group footer for that group.

■ To suppress the group footer,

1 Create a new report using the CUSTOMER table.

2 Select the CUST_ID field, and click the Footer toolbar button to sort and group by that field.

3 Choose Tools |Macro.

4 Select the Report Macros option, enter the name “ConditionalFooter” in the Macro Name box, and then choose New.

The Edit Macro dialog box appears, and ReportSmith inserts the first and last lines of your macro into the formula for you.

5 Enter the following code between the first and last lines of your formula:

```
if Field$("CSTMR_ID") = "C17" then ResumeEvent 0
```

ResumeEvent 0 cancels creation of the “C17” footer object, but only when linked to the Group Footer object’s Creation event.

6 Test your macro, and after you receive a success message, choose OK to return to the Macro Commands dialog box. If you want, choose Save to save the macro to a .MAC file.

▸ Next, you need to link the macro to specific objects in the report,

1 Choose Links.

The Macro Links dialog box appears.

2 Select the following items:

- From the Object Type drop-down box, choose Group Footer. The drop-down box on the right side of the window displays “Groups” as the selection and a list of groups used in the report.
- From the list of groups, select the CUST_ID group.
- From the Events list, select the Creation event (if it is not already selected).

3 Choose Link, and then press OK to return to the Macro Command dialog box.

4 Save the macro to a named *.MAC file.

5 Choose Run to execute the macro in your report. ReportSmith runs the macro and suppresses the Group footer for the “C17” field.

Calling other macros

After you create a macro (A), you can create a second macro (B) that uses the functionality of macro A. You can use parameters to pass information between the two macros. When you do this, ReportSmith replaces the default parameter values in macro A with the parameter values supplied by macro B.

■ To run macro A from macro B, use the *RunMacro* command. The RunMacro command lets you run macro B and supply a string that represents the parameters in macro A.

The RunMacro command

The RunMacro command takes two string parameters:

- The first parameter is the name of a macro to call.
- The second parameter is a list of parameters to pass to the macro you are calling.
  - Data for string-type parameters must be enclosed by double-quotes, so you will usually use the CHR$(34) function to create any double-quote characters that you want to be displayed as part of the string itself. Then, use the string concatenation operator (+) to add the double-quote characters.

Variables can be passed to a macro only *by value* when using the RunMacro command, so the called macro cannot modify the variable value or store a new value in the variable.

Using the DataSet Control

The DataSet Control enables you to define a description of report data and then use that description as a single object for purposes of data manipulation and macro control. For example, you can use the properties and methods of the DataSet Control to make a connection, set a list of tables, set selection criteria, obtain a list of available tables, get a list of table owners, create a default columnar report, and refresh a temporary table for a report.

Creating a DataSet Control Object

You can create a DataSet control with *Dim* and *Global* statements. For example, to create a DataSet control called MyData, your code might look like this:

```
Dim MyData as DataSet
```

Local scope

If you just want to use the dataset control in a single function within the macro, place the Dim statement *inside* that function. This gives the control local scope. It is created when the function is called and destroyed when the function is complete.

Modular scope

If you want all subroutines in a macro to be able to use your control, place the Dim statement *outside* of all subroutines. A macro declared in this way has modular scope. It is available from anywhere within the macro and destroyed when the macro ends.

Global Scope

You can use the Global statement to create a dataset control that is known to all macros in ReportSmith. The Global statement must appear in all modules which refer to the global dataset, and should appear on the first line of your macro before any subroutines or functions.

Properties and methods

The DataSet control is used by employing its properties and methods.

Properties

Properties are attributes that define how dataset objects are displayed and how they function in the running application. They are similar to BASIC variables and can have any of the BASIC variable types. You can access the properties of an object by using the name of the object, followed by a period and the name of the property.

One property of the DataSet control is Selection$. It is a string variable that defines the SQL text following the WHERE clause in an SQL statement. You can use this property to set the selection criteria for a DataSet as in the following example:

```
' Display the Old Selection Criteria.
Msgbox MyData.Selection$                 ◄—— This displays the current SQL selection string...

' Set the New Selection Criteria.
MyData.Selection$ = "SALARY > 40000"     ◄—— ...and this sets “SALARY > 40000” as the new SQL
                                             selection string.
```

Some properties are read-only while others can be both read and written.

Examples of how properties are used:

- Use them in commands: Msgbox MyData.Name$ creates a message box that displays the value of the *Name$* property of the *MyData* dataset control.
- Use them in expressions: Total_Recs = Data1.RecordCount + Data2.RecordCount adds the values of the *RecordCount* property of Data1 and Data2 and then assigns the total to a variable called *Total_Recs*.
- Assign values to them: MyData.Name$ = “My Name” assigns a value of “My Name” (without the double-quotes) to the *Name$* property of the MyData dataset control.

Table 9.1 lists and briefly describes the function of each property. Refer to “Command reference” on page 261 for the syntax, comments and examples of each property.

Table 9.1: DataSet Properties Reference

Property	**Description**
AllDataBases$	Returns all the databases available under the current connection.
AllOwners$	Returns all owners available under the current connection.
AllTables$	Returns a list of all owners for connections that have owners.
DataBases$	Returns the current database for the current connection.
Error$	Contains a string describing the error that occurred in the last dataset control method executed.
Id Property	Used to store an integer value.
Name$	Returns the current name for the current situation.
Owner$	Returns the current owner for the current situation.
Record	Returns the current record in the dataset.
RecordCount	Returns the total number of records in the dataset.
Selection$	Can be used to get or set the selection criteria for a dataset control object.
Tables	Returns a list of tables included in a report.

Methods

Methods are functions or statements that perform actions on the control. The *SetFromActive* method associates the DataSet control with the currently active report. The *Save* method can be used to store a dataset control in a DOS file.

Table 9.2 offers a brief description of each method. Refer to “Command reference” on page 261 for the syntax, parameters, comments and specific examples of each method.

Table 9.2: DataSet Methods Reference

Method	Description
AddGroup	Adds grouping criteria.
AddSort	Adds a sorting criteria.
AddSummary	Adds a summary field.
AddTable	Adds a table.
Connect	Replaces previous connection information with new information.
CreateReport	Allows a DataSet control with a defined query to create one of four different report types.
Disconnect	Removes a connection previously set with the Connect Method.
Field$	Returns the value of the specified data field for the current method.
GetAllField$	Returns a list of fields available for the specified field in the specified table.
GetColumnAlias	Returns the alias for the specified field in the specified table.
GetDataSources	Returns a list of all available data sources.
GetFieldList$	Returns a list of fields in the given table.
GetGroup$	Returns a string providing information about grouping.
GetSort$	Returns a string indicating the sorting criteria at the given level.
GetSQL$	Returns the last SQL statement executed for the dataset control object.
GetSummary$	Returns a string providing information about a summary field.
GetTable$	Returns a string describing the table at the specified index.
GetTableAlias$	Returns the alias for the specified table.
GetTableLink$	Returns a string providing information about the table link.
Include Field$	Set the list of fields that should be included in a table.
LinkMacro	Links a macro in an active list to the specified object, even and item.
Load	Replaces connection information with that specified by the Filename$ parameter.
LoadMacro	Allows a macro to be loaded into the active macro list from a .MAC file.
Recalc	Execute the SQL for a DataSet.
RemoveGroup	Removes a grouping criteria from a report.
RemoveSort	Removes the sorting criteria at the given level.
RemoveSummary	Removes a summary field from a report.
RemoveTable	Removes a table at the specified index.

Table 9.2: DataSet Methods Reference (continued)

Method	Description
RemoveTableLink	Removes the table link at the specified index from the dataset control object.
ReplaceTable	Replaces one table in a report with another.
Save	Store the object in a file.
SetColumnAlias	Sets or changes the Alias for a column in a table.
SetFromActive	Associates the object with the active report.
SetFromLoading	Associates the dataset control object with a report being loaded.
SetTableAlias	Sets or changes the Alias for a table in a report.
SetTableLink	Sets a logical link between two tables that are part of this DataSet.
SetUserSQL	Places the dataset control object into user entered SQL mode.
TestSelection$	Returns a string telling how many records would be selected with the current selection criteria.